University of Glasgow

School of Computing Science

60 YEARS OF COMPUTING AT GLASGOW

UDP

Networked Systems (H)

Lecture 15

Colin Perkins | https://csperkins.org/ | Copyright © 2017 | This work is licensed under the Creative Commons Attribution-NoDerivatives 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nd/4.0/ or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

Lecture Outline

- The UDP protocol and datagram sockets
- Guidelines for writing UDP-based applications
- Example: DNS

Colin Perkins | https://csperkins.org/ | Copyright © 2017

Using UDP Datagrams

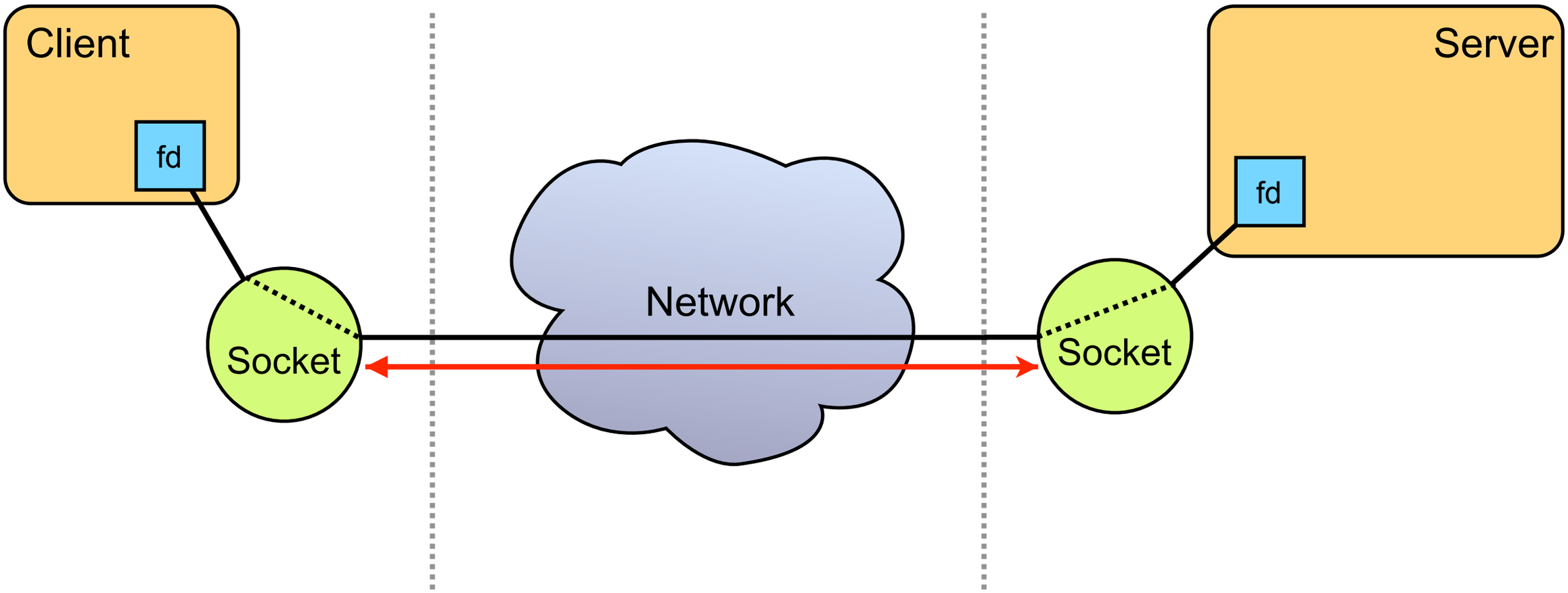
- UDP provides an unreliable datagram service, identifying applications via a 16 bit port number
  - UDP ports are separate from TCP ports
  - Often used peer-to-peer (e.g., for VoIP), so both peers must `bind()` to a known port
  - Create via `socket()` as usual, but specify `SOCK_DGRAM` as the socket type:

```
int     fd;
...
fd = socket(AF_INET, SOCK_DGRAM, 0);
```

- No need to `connect()` or `accept()`, since no connections in UDP

CC BY ND

Using UDP Datagrams




```
int fd = socket(...)
bind(fd, ..., ...)
sendto(fd, data, datalen, addr, addrlen)
recvfrom(fd, buffer, buflen, flags, addr, addrlen)
close(fd)
```

Colin Perkins | https://csperkins.org/ | Copyright © 2017

Sending UDP Datagrams

The `sendto()` call sends a single datagram. Each call to `sendto()` can send to a different address, even though they use the same socket.

```
int                 fd;
char                buffer[...];
int                 buflen = sizeof(buffer);
struct sockaddr_in  addr;
...
if (sendto(fd, buffer, buflen, (struct sockaddr *) addr, sizeof(addr)) < 0) {
    // Error...
}
```

Alternatively, `connect()` to an address, then use `send()` to send the data

There is no connection made by the UDP layer, a `connect()` call only sets the destination address for future packets.

CC BY ND

Receiving UDP Datagrams

The `recv()` call may be used to read a single datagram, but doesn't provide the source address of the datagram. Most code uses `recvfrom()` instead – this fills in the source address of the received datagram:

```
int                 fd;
char                buffer[...];
int                 buflen = sizeof(buffer);
struct sockaddr     addr;
socklen_t           addr_len = sizeof(addr);
int                 rlen;
...
rlen = recvfrom(fd, buffer, buflen, 0, &addr, &addrlen);
if (rlen < 0) {
    // Error...
}
```

CC BY ND

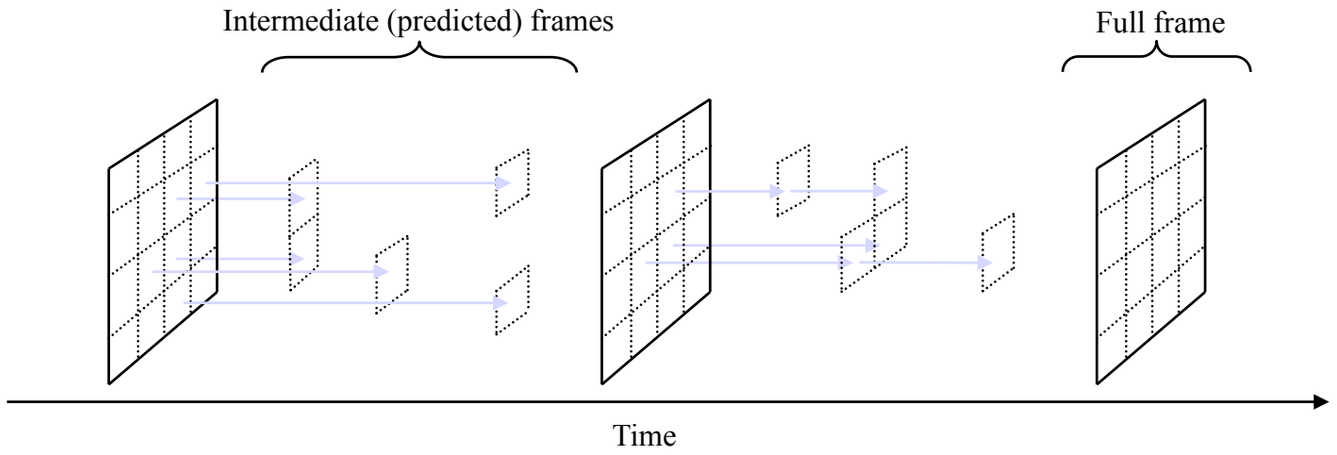
UDP Framing and Reliability (1)

- Unlike TCP, each UDP datagram is sent as exactly one IP packet (which may be fragmented in IPv4)
  - Each `recvfrom()` corresponds to a single `sendto()`
- But, transmission is unreliable: packets may be lost, delayed, reordered, or duplicated in transit
- The application is responsible for correcting the order, detecting duplicates, and repairing loss – if necessary
- Generally requires the sender to include some form of sequence number in each packet sent

CC BY ND

UDP Framing and Reliability (2)

- UDP provides framing – data is delivered a packet at a time – but is unreliable
- Application must organise the data so it's useful if some packets lost
  - E.g. streaming video with I and P frames

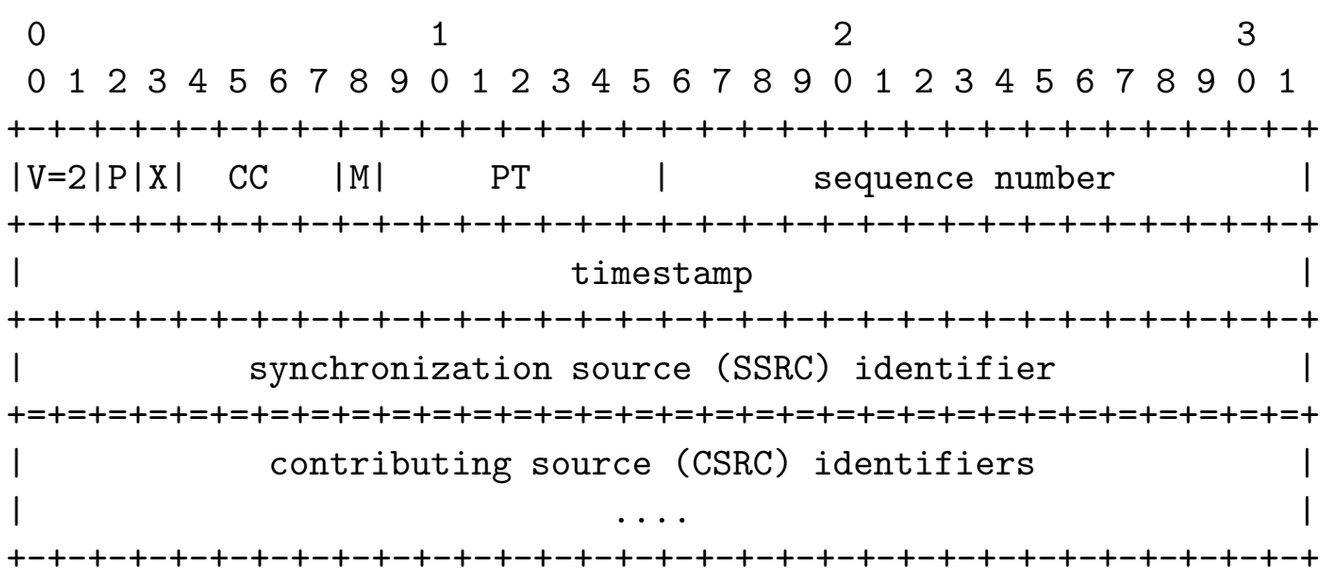


D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. Proc SIGCOMM Conference, pages 200–208, Philadelphia, PA, USA, September 1990. ACM. http://dx.doi.org/10.1145/99517.99553

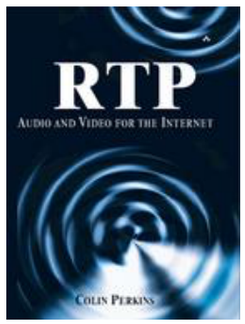
Colin Perkins | https://csperkins.org/ | Copyright © 2017

Sequencing and Reliability

- Need to provide sequencing, reliability, and timing in applications
  - Sequence numbers and acknowledgements
  - Retransmission and/or forward error correction
  - Timing recovery
- Example: RTP data packet header



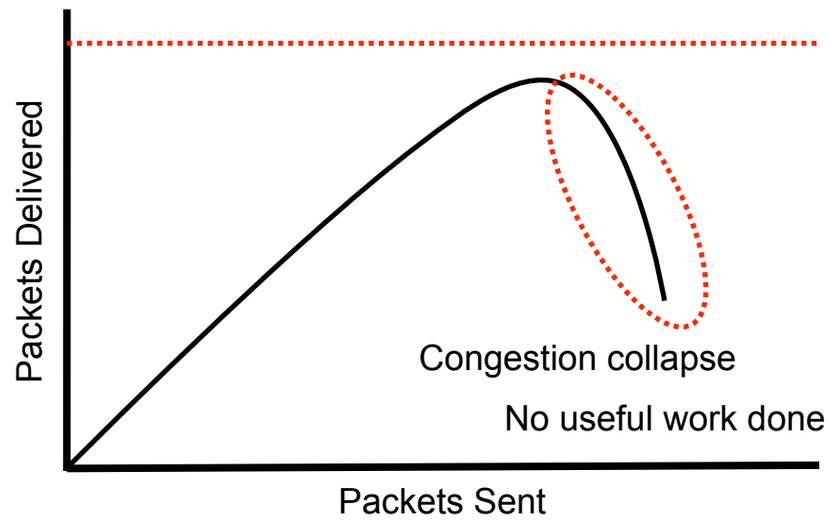
See RFC 3550 or



CC BY ND

Congestion Control

- Need to implement congestion control in applications
  - To avoid congestion collapse of the network
  - Should be approximately fair to TCP



- Difficult to do well – TCP congestion control covers many corner-cases that are easy to miss
  - RFC 3448 provides a detailed specification for a well-tested algorithm
  - IETF RMCAT working group developing standard congestion control algorithms for interactive video applications running over UDP https://datatracker.ietf.org/wg/rmcat/charter/
  - Google's QUIC protocol builds on UDP to give more sophisticated service

CC BY ND

Guidelines for writing UDP applications

- IETF provides guidelines for writing UDP-based applications
  - RFC 5405 – https://tools.ietf.org/html/rfc5405
  - Under revision: https://tools.ietf.org/html/draft-ietf-tsvwg-rfc5405bis-19

- Read this before trying to write UDP-based code

CC BY ND

Example: Domain Name System

- The most widely used UDP-based application is the domain name system (DNS)
  - The network operates entirely on IP addresses, and has no concept of names for hosts
  - The DNS is an application that translates from user-visible names to IP address
    - www.dcs.gla.ac.uk → 130.209.240.1
  - DNS is an application layer protocol, running over the network
  - Not necessary for the correct operation of the transport or network layers, or lower
- Why run over UDP? Request-response protocol, where it was thought TCP connection setup and congestion control was too much overhead
  - Unclear if this design is appropriate

CC BY ND

History of the DNS

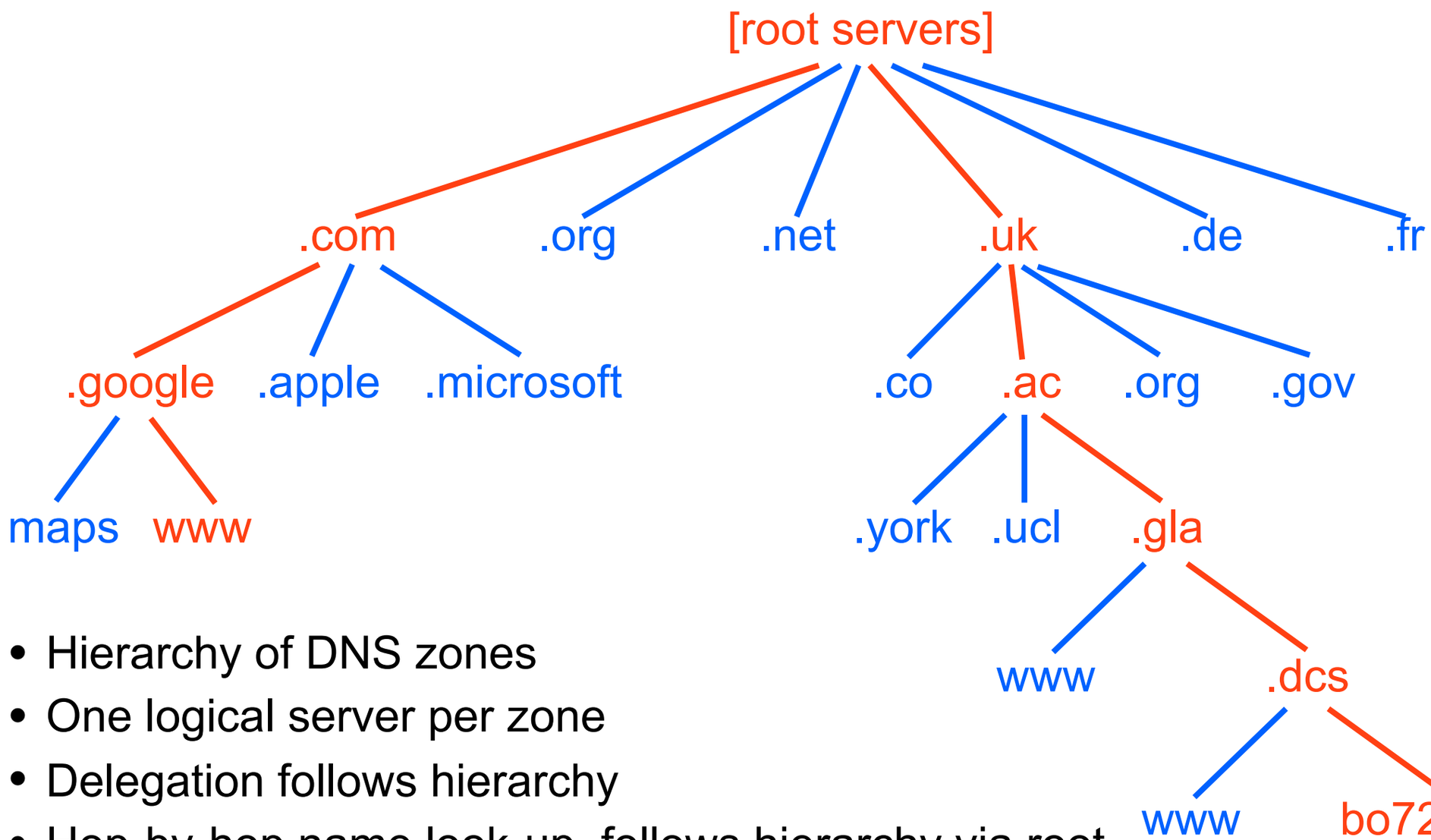
- Early Internet didn't use DNS
  - Flat file hosts.txt listing all host names and addresses
  - Maintained by central NIC; updated by email every few days; manually installed in hosts
- DNS proposed in 1983 as distributed database of host names
- Solve scaling problems with hosts.txt



Paul Mockapetris

CC BY ND

Operation of the DNS



- Hierarchy of DNS zones
- One logical server per zone
- Delegation follows hierarchy
- Hop-by-hop name look-up, follows hierarchy via root
- Results have TTL, cached at intermediate servers
- getaddrinfo()

CC BY ND

Contents of a DNS Zone

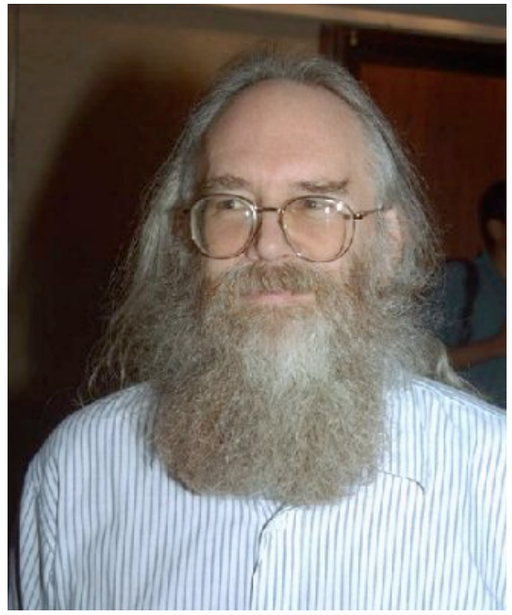
```
$TTL 3600          ; 1 hour
example.org.    IN      SOA     ns1.example.org. admin.example.org. (
                                2006051501      ; Serial
                                10800           ; Refresh
                                3600            ; Retry
                                604800          ; Expire
                                86400           ; Minimum TTL
                        )
; DNS Servers
                IN      NS      ns1.example.org.
                IN      NS      ns2.example.org.
; MX Records
                IN      MX 10   mx.example.org.
                IN      MX 20   mail.example.org.
; Machine Names
ns1             IN      A       192.168.1.2
ns2             IN      A       192.168.1.3
mx              IN      A       192.168.1.4
mail            IN      A       192.168.1.5
mail            IN      AAAA    2001:200:1000:0:25f:23ff:fe80:1234
server1         IN      A       192.168.1.10
server2         IN      A       192.168.1.11
; Aliases
www             IN      CNAME   server1
```

Source: adapted from The FreeBSD Handbook

CC BY ND

DNS Politics

- The DNS was administered by IANA
  - Jon Postel *was* IANA from its creation until his death in 1998
  - http://www.ietf.org/rfc/rfc2468.txt – “I remember IANA”
- DNS now managed into ICANN
  - The US government asserts ultimate control over ICANN, and hence the DNS
  - Significant attempts to move control of national domains to the UN, and hence to the countries concerned
  - Other attempts to set up alternate roots for the DNS, with different namespaces → significant technical problems



Jon Postel

CC BY ND

Summary

- UDP and datagram sockets
- Guidelines for using UDP
- A UDP-based application: DNS

Colin Perkins | https://csperkins.org/ | Copyright © 2017